**Q.1 a. What is a Turing test and what capabilities are required by a computer to pass this test?**

**Answer:**

The Turing Test was designed to provide a satisfactory operational definition of intelligence. A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer.

The computer would need to possess the following capabilities:

* natural language processing to enable it to communicate successfully in English;

* knowledge representation to store what it knows or hears;

* automated reasoning to use the stored information to answer questions and to draw new conclusions;

* machine learning to adapt to new circumstances and to detect and extrapolate patterns

**b. Differentiate tree based breadth-first, depth-first and iterative-deepening search strategies based on completeness, time and space complexities.**

**Answer:**

| Criterion | Breadth-First | Uniform-Cost | Depth-First | Depth-Limited | Iterative Deepening | Bidirectional (if applicable) |
|---|---|---|---|---|---|---|
| Complete? | Yes[a] | Yes[a,b] | No | No | Yes[a] | Yes[a,d] |
| Time | $O(b^d)$ | $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ | $O(b^m)$ | $O(b^\ell)$ | $O(b^d)$ | $O(b^{d/2})$ |
| Space | $O(b^d)$ | $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ | $O(bm)$ | $O(b\ell)$ | $O(bd)$ | $O(b^{d/2})$ |
| Optimal? | Yes[c] | Yes | No | No | Yes[c] | Yes[a,d] |

**c. Define minimax value with regards to optimal decision in games**

**Answer:**

Given a game tree, the optimal strategy can he determined from the minimax value of each node, which we write as MINIMAX(n). The minimax value of a node is the utility (for MAX) of being in the corresponding state, assuming that both players play optimally from there to the end of the game. Obviously, the minimax value of a terminal state is just its utility. Furthermore, given a choice, MAX prefers to move to a state of maximum value, whereas MIN prefers a state of minimwn value.

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

**d. Provide an algorithm for converting an expression into prenex normal form.**

**Answer:**

**Algorithm for prenex normal form**

Any expression can be converted into prenex normal form. To do this, the following steps are needed:

1. Eliminate all occurrences of → and ↔ from the formula in question.

2. Move all negations inward such that, in the end, negations only appear as part of literals.

3. Standardize the variables apart (when necessary).

4. The prenex normal form can now be obtained by moving all quantifiers to the front of the formula.

**e. List merits and demerits of frames in Knowledge Representation**

**Answer:**

Merits:

Makes programming easier by grouping related knowledge

- Easily understood by non-developers
- Expressive power
- Easy to set up slots for new properties and relations

Easy to include default information and detect missing values

Demerits:

- No standards (slot-filler values)
- More of a general methodology than a specific representation. For example, a frame for a class-room will be different for a professor and for a maintenance worker
- No associated reasoning/inference mechanisms

**f. Consider a doctor diagnosing a rare form of bowel syndrome. He knows that only 0.1% of the population suffers from that disease. He also knows that if a person has the disease, the test has 99% chance of turning out positive. If the person doesn't have the disease, the test has a 98% chance of turning negative. How feasible is this diagnostics method? That is, given that a test turned out positive, what are the chances of the person really having the disease?**

**Answer:**

Let's say that event DIS is having the disease, and event POS is getting a positive test. To solve the problem we want to find P(DIS|POS).

P(DIS) = 0.001 (population with the disease)

P(DIS') = 0.999 (population without the disease, so complement of event DIS)

P(POS|DIS) = 0.99 (positive test given patient has disease)

P(POS|DIS') = 0.01 (positive test given patient doesn't have disease)

P(POS'|DIS) = 0.02 (negative test given patient has disease)

P(POS'|DIS') = 0.98 (negative test given patient doesn't have the disease)

P(POS) = P(DIS)P(POS|DIS) + P(DIS')P(POS|DIS')

P(POS) = 0.001 * 0.99 + 0.999 * 0.02

P(POS) = 0.02097

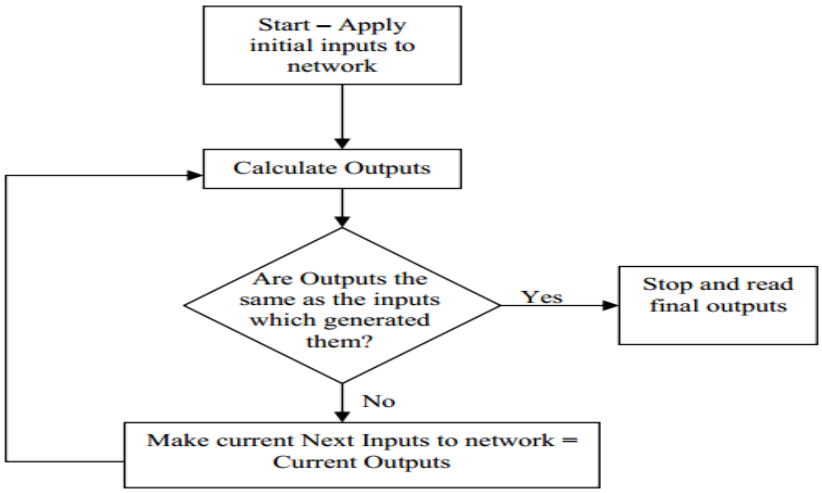
P(DIS|POS) = 0.99 * 0.001 / 0.02097 = 0.0472103

Given the test turned out to be positive, the person only has a chance of 4.7% of actually having the disease. So this is not a feasible method for diagnosing the rare disease.

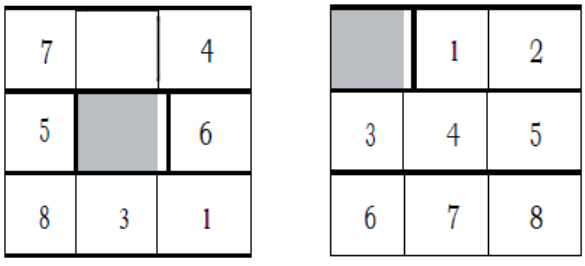
**g. What is a recurrent network? Give the basic flow for recurrent networks. (7 ×4)**

**Answer:**

Adding feedback connections to the network (the outputs are fed back into the inputs) results in a recurrent network. Networks with such connections are called “feedback” or “recurrent” networks. Previous outputs are fed as current inputs. This is done until the outputs don’t change any more (they remain constant). At this point the network is said to have relaxed.




**Q.2 a. Consider the 8-piece sliding block problem, shown in the Figure 1. Write the standard problem formulation. (6)**




Start State Goal State

**Answer:**

The 8-puzzle consists of a 3x3 board with eight numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The object is to reach a specified goal state. The standard formulation is as follows:

• States: A state description specifies the location of each of the eight tiles and the blank

in one of the nine squares.

• Initial state: Any state can be designated as the initial state. Note that any given goal

can be reached from exactly half of the possible in initial states

• Actions: The simplest formulation defines the actions as movements of the blank space

Left, Right, Up, or Down. Different subsets of these are possible depending on where

the blank is.

• Transition model: Given a state and action, this returns the resulting state.

• Goal test: This checks whether the state matches the goal configuration

• Path cost: Each step costs 1, so the path cost is the number ofsteps in the path.

**b. Explain local beam search with example in detail. (6)**

**Answer:**

Keeping just one node in memory might seem to be an extreme reaction to the problem of memory limitations. The local beam search algorithm keeps track of k states rather than just one. It begins with k randomly generated states. At each step, all the successors of all k states are generated. If any one is a goal, the algorithm halts. Otherwise, it selects the k best successors from the complete list and repeats.

At first sight, a local beam search with k states might seem to be nothing more than mnning k random restarts in parallel instead of in sequence. In fact, the two algorithms are quite different. In a random-restart search, each search process runs independently of the others. In a local beam search, useful information is passed among the parallel search threads. In effect, the states that generate the best successors say to the others, "Come over here, the grass is greener!" The algorithm quickly abandons unfruitful searches and moves its resources to where the most progress is being made.

In its simplest form, local beam search can suffer from a lack of diversity among the k states-they can quickly become concentrated in a small region of the state space, making the search little more than an expensive version of hill climbing. A variant called stochastic beam search, analogous to stochastic hill climbing, helps alleviate this problem. Instead of choosing the best k from the pool of candidate successors, stochastic beam search chooses k successors at random, with the probability of choosing a given successor being an increasing function of its value. Stochastic beam search bears some resemblance to the process of natural selection, whereby the "successors" (offspring) of a "state" (organism) populate the next generation according to its "value" (fitness).

**c. How does iterative deepening help? For search in binary tree, show four iterations using iterative deepening. (6)**

**Answer:**

Iterative deepening search (or iterative deepening depth-first search) is a general strategy,often used in combination with depth-first tree search, that finds the best depth limit. It doesthis by gradually increasing the limit-first 0, then 1, then 2, and so on-until a goal is found.

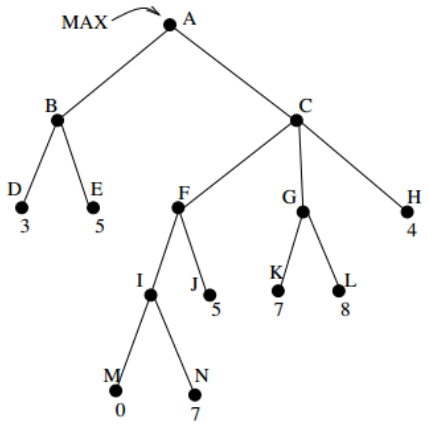
This will occur when the depth limit reaches d, the depth of the shallowest goal node.

Iterative deepening combines the benefits of depth-firstand breadth-first search. Like depth-first search, its memory requirements are modest: O(bd)to be precise. Like breadth-first search, it is complete when the branching factor is finite andoptimal when the path cost is a non decreasing function of the depth of the node.

Figure to be given for four iterations of ITERATIVE-DEEPENING-SEARCH on a binary search tree, where thesolution is found on the fourth iteration.

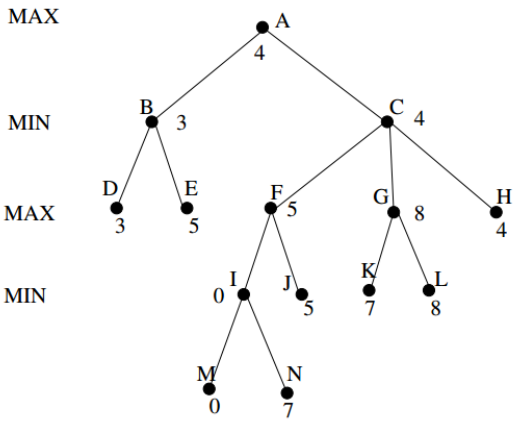
Iterative deepening search may seem wasteful because states are generated multipletimes. It turns out this is not too costly. The reason is that in a search tree with the same (ornearly the same) branching factor at each level, most of the nodes are in the bottom level,so it does not matter much that the upper levels are generated multiple times. In an iterativedeepening search, the nodes on the bottom level (depth d) are generated once, those on the next-to-bottom level are generated twice, and so on, up to the children of the root, which aregenerated d times.

**Q.3 Consider the following game tree.**




**a. Find the best move for the MAX player using the minimax procedure. (6)**

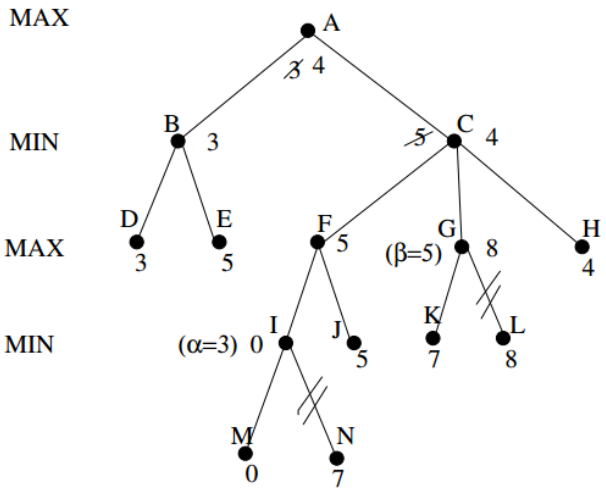
**Answer:**




**b. Perform a left-to-right alpha-beta pruning on the tree. Indicate where the cutoffs occur. (6)**

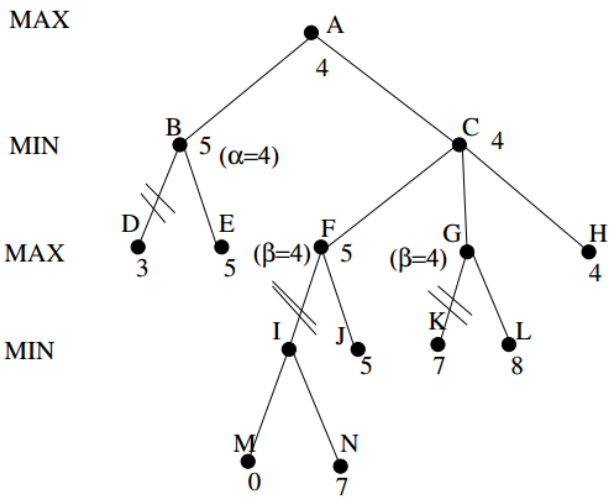
**Answer:**

Alpha-beta pruning left to right




**c. Perform a right-to-left alpha-beta pruning on the tree. Discuss why different pruning occurs. (6)**

**Answer:**




**Q.4 a. In how many ways can a group of 9 people work in 3 disjoint subgroups of 2, 3 and 4 persons?**
**Example:**
**?- group3([aldo,beat,carla,david,evi,flip,gary,hugo,ida],G1,G2,G3).**
**G1 = [aldo,beat], G2 = [carla,david,evi], G3 = [flip,gary,hugo,ida]**
**...**
**Write a predicate using prolog that generates all the possibilities. (9)**

**Answer:**

```
        % group3(G,G1,G2,G3) :- distribute the 9 elements of G into G1, G2,
and G3,
%    such that G1, G2 and G3 contain 2,3 and 4 elements respectively

group3(G,G1,G2,G3) :-
   selectN(2,G,G1),
   subtract(G,G1,R1),
   selectN(3,R1,G2),
   subtract(R1,G2,R2),
   selectN(4,R2,G3),
   subtract(R2,G3,[]).

% selectN(N,L,S) :- select N elements of the list L and put them in
%    the set S. Via backtracking return all posssible selections, but
%    avoid permutations; i.e. after generating S = [a,b,c] do not return
%    S = [b,a,c], etc.

selectN(0,_,[]) :- !.
selectN(N,L,[X|S]) :- N > 0,
   el(X,L,R),
   N1 is N-1,
   selectN(N1,R,S).

el(X,[X|L],L).
el(X,[_|L],R) :- el(X,L,R).

% subtract/3 is predefined
```

**b. Explain simple planning using goal stack with algorithm. (9)**

**Answer:**

One of the earliest techniques is planning using goal stack. Problem solver uses single stack that contains sub goals and operators bothsub goals are solved linearly and then finally the conjoined sub goal is solved.Plans generated by this method will contain complete sequence of operations for solving one goal followed by complete sequence of operations for the next etc.Problem solver also relies on a database that describes the current situation.Set of operators with precondition, add and delete lists.

Algorithm:

Let us assume that the goal to be satisfied is:

GOAL = G1 Λ G2 Λ … ΛGn

Sub-goals G1, G2, … Gn are stacked with compound goal G1 Λ G2 Λ …Λ Gn at the bottom.

Top → G1
G2
:
Gn
Bottom → G1 Λ G2 Λ … Λ G4

At each step of problem solving process, the top goal on the stack is pursued.

- Find an operator that satisfies sub goal G1 (makes it true) and replace G1 by the operator.
  - If more than one operator satisfies the sub goal then apply some heuristic to choose one.
- In order to execute the top most operation, its preconditions are added onto the stack.
  - Once preconditions of an operator are satisfied, then we are guaranteed that operator can be applied to produce a new state.
  - New state is obtained by using ADD and DELETE lists of an operator to the existing database.
- Problem solver keeps tract of operators applied.
  - This process is continued till the goal stack is empty and problem solver returns the plan of the problem

**Q.5 a. Consider the statements in the following paragraph:**
**"Every human, animal and bird is living thing who breathe and eat. All birds can fly. All man and woman are humans who have two legs. Cat is an animal and has a fur. All animals have skin and can move. Giraffe is an animal who is tall and has long legs. Parrot is a bird and is green in color"**
**For the above paragraph, provide predicate logic and semantic net representations. (9)**

**Answer:**

Predicate logic:

Every human, animal and bird is living thing who breathe and eat.

**∀X [human(X) → living(X)]**

**∀X [animal(X) → living(X)]**

**∀X [bird(X) → living(X)]**

All birds are animal and can fly.

**∀X [bird(X) ∧ canfly(X)]**

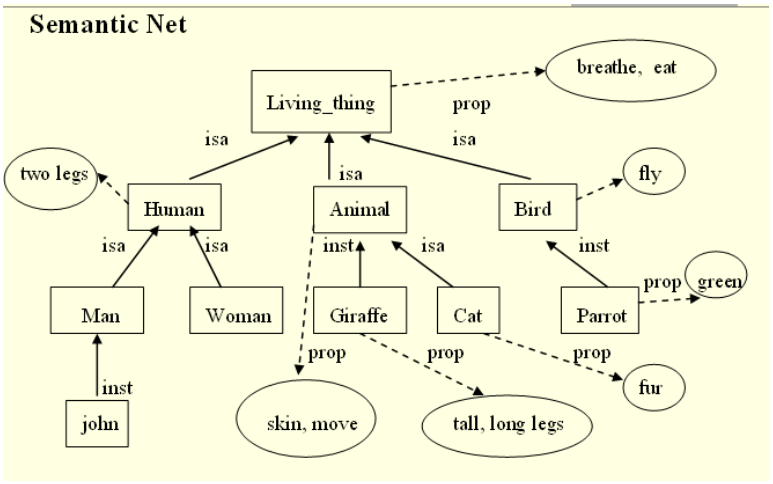
Every man and woman are humans who have two legs.

**∀X [man(X) ∧ haslegs(X)]**

**∀X [woman(X) ∧ haslegs(X)]**

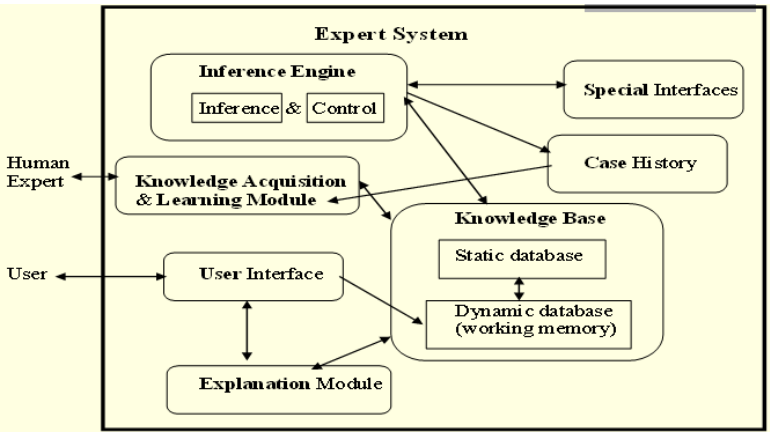
**∀X [human(X) ∧ has(X, legs)]**
Cat is an animal and has a fur.
**animal(cat) ∧ has(cat, fur)**
All animals have skin and can move.
**∀X [animal(X) → has(X, skin) ∧ canmove(X)]**
Giraffe is an animal who is tall and has long legs.
**animal(giraffe) ∧ has(giraffe, long_legs) ∧ is(giraffe, tall)**
Parrot is a bird and is green in color.
**bird(parrot) ∧ has(parrot, green_colour)**

Semantic Net:




**b. Provide and explain Expert System (ES) architecture illustrating role of every component. How can prolog be used with forward chaining as control strategy? (9)**

**Answer:**




Architecture components details:

**Knowledge base (KB)**

KB consists of knowledge about problem domain in the form of static and dynamic databases. Static knowledge consists of rules and facts which is complied as a part of the system and does not change during execution of the system. Dynamic knowledge consists of facts related to a particular consultation of the system. At the beginning of the consultation, the dynamic

knowledge base often called working memory is empty. As a consultation progresses, dynamic knowledge base grows and is used along with static knowledge in decision making. Working memory is deleted at the end of consultation of the system

**Inference Engine**

It consists of inference mechanism and control strategy. Inference means search through knowledge base and derive new knowledge. It involve formal reasoning involving matching and unification similar to the one performed by human expert to solve problems in a specific area of knowledge. Inference operates by using modus ponen rule. Controlstrategy determines the order in which rules are applied. There are mainly two types of control mechanism viz., forward chaining and backward chaining.

**Knowledge Acquisition**

Knowledge acquisition module allows system to acquire knowledge about the problem domain. Sources of Knowledge for ES include text books, reports, case studies, empirical data and domain expert experience. Updation of Knowledge can be done using knowledge acquisition module of the system. Insertion, deletion and updation of existing knowledge are all possible

**Case History**

Case History stores the file created by inference engine using the dynamic database created at the time of consultation. Useful for learning module to enrich its knowledge base. Different cases with solutions are stored in Case Base system.These cases are used for solving problem using Case Base Reasoning (CBR).

**Prolog in forward chaining:**

Prolog uses backward chaining as a control strategy, but forward chaining can be implemented in Prolog. In forward chaining, the facts from static and dynamic knowledge bases are taken and are used to test the rules through the process of unification. The rule is said to be fired and the conclusion (head of the rule) is added to the dynamic database when a rule succeeds. Prolog rules are coded as facts with two arguments, first argument be left side of rule and second is the list of sub goals in the right side of the rule.

**Q.6 a. Explain Dempster–Shafer Theory with formalism, with an appropriate example (9)**

**Answer:**

It is a mathematical theory of evidence. It allows one to combine evidence from different sources and arrive at a degree of belief. Belief function is basically a generalization of the Bayesian theory of probability.Belief functions allow us to base degrees of belief or confidence for one event on probabilities of related events, whereas Bayesian theory requires probabilities for each event.These degrees of belief may or may not have the mathematical properties of probabilities.The difference between them will depend on how closely the two events are related. It also uses numbers in the range [0, 1] to indicate amount of belief in a hypothesis for a given piece of evidence. Degree of belief in a statement depends upon the number of answers to the related questions containing the statement and the probability of each answer.In this formalism, a degree of belief (also referred to as a mass) is represented as a belief function rather than a Bayesian probability distribution

Example to be given

**Formalism**

- Let *U* be the *universal set* of all hypotheses, propositions, or statements under consideration.
    - The power set P(U), is the set of all possible subsets of *U*, including the empty set represented by $\phi$.

- The theory of evidence assigns a belief mass to each subset of the power set.

- A function m: P(U) → [0,1] is called a *basic belief assignment* (BBA) function. It satisfies the following axioms:
  - $m(\phi) = 0$ ; $\sum m(A) = 1$, $\forall A \in P(U)$
- The value of m(A) is called *mass assigned to A* on the unit interval.
- It makes no additional claims about any subsets of A, each of which has, by definition, its own mass.
- The original combination rule, known as Dempster's rule of combination, is a generalization of Bayes' rule.
- Assume that *m1* and *m2* are two belief functions used for representing multiple sources of evidences for two different hypotheses.
- Let A, B $\subseteq$ U, such that m1(A) $\neq$ 0, and m2(B) $\neq$ 0.
- The Dempster's rule for combining two belief functions to generate an *m3* function may be defined as:

$$\mathbf{m3(\phi) = 0}$$

$$\mathbf{m3(C) = \frac{\sum_{A \cap B = C} (m1(A) * m2(B))}{1 - \sum_{A \cap B = \phi} (m1(A) * m2(B))}}$$

- This belief function gives new value when applied on the set C = A ∩ B.
- The combination of two belief functions is called the *joint mass*.
  - Here m3 can also be written as (m1 o m2).
- The expression [ $\sum_{A \cap B = \phi}$ (m1(A) * m2(B))] is called normalization factor.
  - It is a measure of the amount of conflict between the two mass sets.
- The normalization factor has the effect of completely ignoring conflict and attributing any mass associated with conflict to the null set.

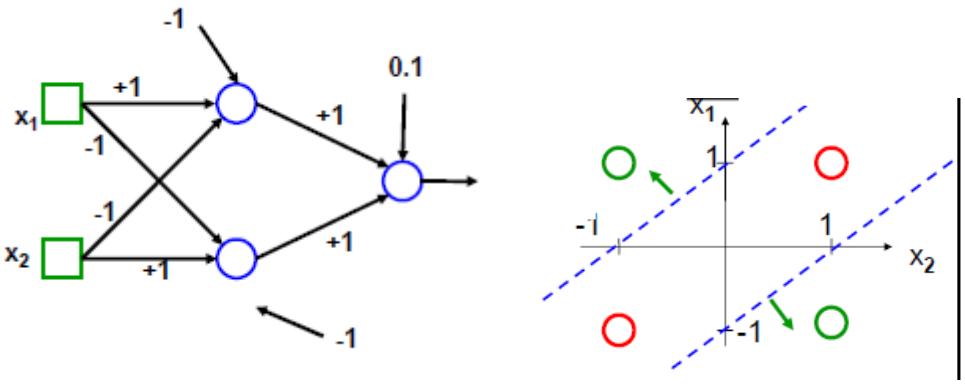
**b. Provide general architecture for multi-layer feed forward neural network. Consider the following truth table non linearly separable XOR function with argument values {-1,1}.**

| $x_1$ | $x_2$ | $x_1$ XOR $x_2$ |
|---|---|---|
| -1 | -1 | **-1** |
| -1 | 1 | **1** |
| 1 | -1 | **1** |
| 1 | 1 | **-1** |

**Provide a multi-layer feed forward neural network to realize the above non-linear separation. (9)**

**Answer:**

In this graph of the XOR, input pairs giving output equal to 1 and -1 are shown. These two classes cannot be separated using a line. We have to use two lines. The following NN with two hidden nodes realizes this non-linear separation, where each hidden node describes one of the two lines.




This NN uses the sign activation function. The two arrows indicate the regions where the network output will be 1. The output node is used to combine the outputs of the two hidden nodes.

**Q.7 a. Consider the Santa Fe Trail problem: The objective of this problem is to evolve a program which eats all the food on a path, without searching too much when there are gaps in the path. A sensor can see the next cell in the direction it is facing. Consider the terminals to be move, left, right: move moves forward one cell, left takes the program to immediate left cell and right takes the program to immediate right cell. Justify why Genetic programming may be an appropriate approach for solving this problem and Provide a suitable fitness function. (4)**

**Answer:**

Problem areas involving many variables that are interrelated in a non-linear or unknown way and an approximate answer is sufficient. The best answer need not be found always. GP approach can greatly reduce time in searching adequate solution.

A suitable fitness function for this can be amount of food collected in 100 time steps.

**b. Compare Genetic Algorithm with traditional optimizing algorithmic approaches in detail. (7)**

**Answer:**

Genetic algorithms differ from conventional optimization and search procedures in several fundamental ways. It can be summarized as follows:

1. Genetic algorithms work with a coding of solution set, not the solutions themselves.
2. Genetic algorithms search from a population of solutions, not a single solution.
3. Genetic algorithms use payoff information (fitness function), not derivatives or other auxiliary knowledge.
4. Genetic algorithms use probabilistic transition rules, not deterministic rules.

**c. List and justify possible applications of Genetic Algorithms. (7)**

**Answer:**

- Optimization: GAs have been used in a wide variety of optimization tasks, including numerical optimization, and combinatorial optimization problems such as traveling salesman problem (TSP), circuit design, job shop scheduling and video & sound quality optimization.
- Automatic Programming: GAs have been used to evolve computer programs for specific tasks, and to design other computational structures, for example, cellular automata and sorting networks.

- Machine and robot learning: GAs have been used for many machine- learning applications, including classification and prediction, and protein structure prediction. GAs have also been used to design neural networks, to evolve rules for learning classifier systems or symbolic production systems, and to design and control robots.
- Economic models: GAs have been used to model processes of innovation, the development of bidding strategies, and the emergence of economic markets.
- Immune system models: GAs have been used to model various aspects of the natural immune system, including somatic mutation during an individuals lifetime and the discovery of multi-gene families during evolutionary time.
- Ecological models: GAs have been used to model ecological phenomena such as biological arms races, host-parasite co-evolutions, symbiosis and resource flow in ecologies.
- Population genetics models: GAs have been used to study questions in population genetics, such as "under what conditions will a gene for recombination be evolutionarily viable?" Interactions between evolution and learning: GAs have been used to study how individual learning and species evolution affect one another.
- Models of social systems: GAs have been used to study evolutionary aspects of social systems, such as the evolution of cooperation, the evolution of communication, and trail-following behavior in ants.
- Areas where large computerized databases are accumulating and computerized techniques are needed to analyze the dat

## Text Book

1. **Elaine Rich and Kevin Knight, Artificial Intelligence, Tata McGraw-Hill, Reprint 2003.**
2. **S Russell and Peter Norvig, Artificial Intelligence-A Modern Approach, Pearson Education , Reprint 2003**
3. **Saroj Kaushik, Logic and Prolog Programming, New Age International Ltd, Publisher, 2007.**